International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 6, June - 2013

Dynamic Resource Management Using Mapreduce Framework In Heterogeneous Cloud Environment

Mrs. J. Keerthika[#1], Mr. Castro[*2]

[#1]*II M.E.CSE,* [*2]*Asst. Prof, M.E CSE*
ASL Pauls College of Engineering & Technology, Elur Pirivu, Coimbatore, India

Abstract

CLOUD computing shows a work of fiction way to supplement the current utilization and delivery model for IT services based on the Internet, by providing for dynamically scalable and often virtualized resources as a service over the Internet. We tackle the problem of dynamic resource management for a large-scale cloud environment. We honour the resource Share problem as that of dynamically maximizing the cloud utility under CPU and memory Constriction. We extend gossip protocol and mapreduce framework to provide an efficient heuristic solution for the complete problem, which includes minimizing the cost for dynamic adapting an Share. The protocol continuously executes on dynamic, local input and does not require global synchronization. We propose architecture to allocate resources to a MapReduce cluster in the Cloud. In MapReduce, the Map function process the input in the form of key/value pairs to generate intermediate key/value pairs, and the Reduce function process all intermediate values associated with the same intermediate key generated by the Map function. In heterogeneous environments where the time to process a task varies depending on nodes, it is sometimes better to pick a remote task that runs faster on the node. Hence, we need to consider both the time required to read and the time needed to process the data, to pick the best task for the node. We achieve a metric of share in a heterogeneous cluster to realize a scheduling scheme that achieves high Concert and Fairness.

Key Words—Cloud computing, distributed management, Re-source Share, gossip protocols.

1. Introduction

Cloud computing environments provide a delusion of infinite computing resources to cloud users so that they can increase or decrease their resource consumption rate according to the demands. Cloud Environment includes the physical infrastructure and related control functionality that enables the provisioning and management of cloud services in figure 1. In cloud computing environments, there are two players: cloud providers and cloud users. On one hand, providers hold massive computing resources in their large datacenters and rent resources out to users on a per-usage basis. On the other hand, there are users who have applications with actuating
Loads and lease resources from providers to run their applications.While our contribution, we conduct the discussion from the acuity of the Platform-as-a-Service (PaaS) notion, with the specific use case of a cloud service provider which Hosts sites in a cloud Environment. It offers hosting services to site demand owners through a middleware that executes on its infrastructure. Site Owners give services to their respective users via sites that are hosted by the cloud service provider. Our contribution can also be applied to the Infrastructure-as-a-Service (IaaS) notion. A use case for this notion could include a cloud tenant running a collection of virtual appliances that are hosted on the cloud infrastructure, with services provided to end users through the public Internet.

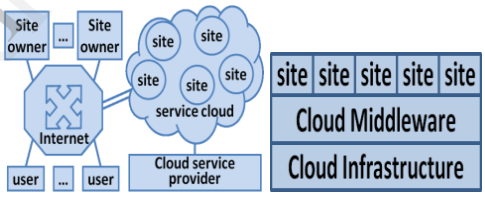

Figure 1.Cloud architecture

The intend goal of cloud Environment:

- Concert objective: Objective is to achieve maxmin sprite among sites for computational resources under memory Constriction.
- Adaptability: The resource Share process must with dynamic and resourcefully adapt to changes in the demand from sites.
- Scalability: To achieve scalability, we envision that all key tasks of the middleware layer, including estimating global states, placing site modules and compute policies for request forwarding are based on distributed algorithms.

These solutions include functions that compute placements of applications or virtual machines onto specific physical machines. However, in a combined and integrated form, (a) with dynamism adapt existing placements in response to a change (b) with dynamism scale resources for an application beyond a single physical machine, (c)scale beyond some thousand physical machines. These three features in integrated form characterize our contribution. The notions in this

paper thus outline a way to improve placement functions in these solutions.

2. System architecture

Datacenters running a cloud Environment often contain a large number of machines that are linked by a high-speed network. Users access sites hosted by the cloud Environment through the public Internet. A site is typically accessed through a URL that is translated to a network address through a global directory service, such as DNS. Figure 2 (left) shows the architecture of the cloud middleware. The mechanism of the middleware layer runs on all machines. Each machine runs a *machine manager* component that computes the resource Share policy, which includes deciding the module instances to run. The resource Share policy is computed by a protocol that runs in the *resource manager* component. This component takes as input the estimated demand for each module that the machine runs. The computed Share policy is sent to the *module scheduler* for implementation/execution, as well as the *site managers* for making decisions on request forwarding. The *overlay manager* implements a distributed algorithm that maintains an overlay graph of the machines in the cloud and provides each resource manager with a list of machines to interact with.

Our architecture associates one site manager with each site. A site manager handles user requests to a particular site. It has two mechanisms: a *demand profiler* and a *request forwarder*. The *demand profiler* estimates the resource demand of each module of the site based on request statistics, QoS targets, etc. Similarly, the *request forwarder* sends user requests for processing to instances of modules belonging to this site.

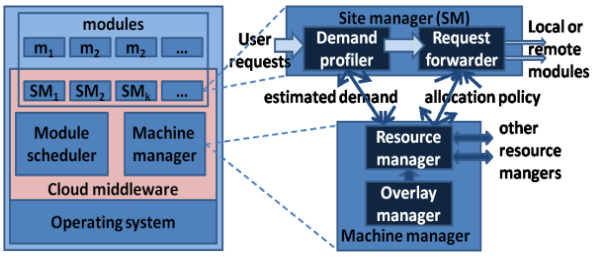

Figure 2.The architecture for the cloud middleware (left) and mechanism for request handling and resource Share (right).

Figure 2 (right) shows the mechanism of a site manager and how they relate to machine managers. The above architecture is not appropriate for the case where a single site manager can't handle the incoming request stream for a site.

3. Problem statement

In the existing system, it focus on homogeneous environments. In a homogeneous environment where all the machines have the same computing capacity. Here we consider data locality and network connectivity when making a job scheduling decision. If multiple jobs receive their fair share of resources, it is enough to count the number of machines assigned to each job. The same applies when the user makes a resource request; him or her only need to specify the number of machines wants. Drawback is cloud that spans a single datacenter containing a single cluster of machines and efficiency will be stumpy. We proposed heterogeneous environment to schedule a job on its preferred allocate resources to achieve high Concert and Fairness using MapReduce cluster in the cloud.

4. Module description

4.1. Resource share by cloud middleware

We present gossip protocol for resource Share in a cloud Environment as P*. P* has the structure of a round based distributed algorithm. Node interaction with P* follows the so-called push-pull archetype, whereby two nodes exchange state information, process this information and update their local states during a round. P* runs on all machines of the cloud. After that, it invokes P* to compute and with dynamism adapt the configuration with the goal to optimize the cloud utility. Here, we consider a cloud as having computational resources and memory resources, which are available on the machines in the cloud infrastructure. The protocol P* takes as input the available cloud resources, the current configuration *A* and the current resource demand. The specific problem we address is that of placing modules on machines and allocating cloud resources to these modules, such that a cloud utility is maximized under Constriction. OP(1) optimized the resource Share with memory Constriction.

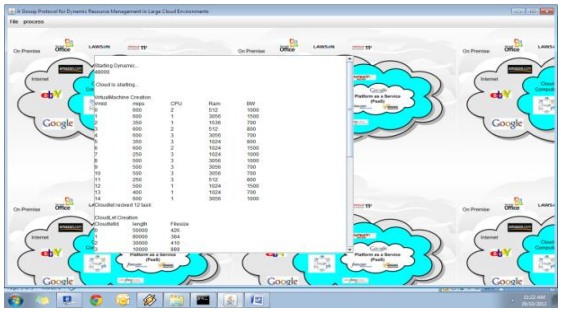

Figure 3. Resource Share to Virtual Machine and Host by gossip protocol in cloud

We model the cloud as a system in Figure 3 with a set of sites *S* and a set of machines *N* that run the sites. Each site *s* ∈*S* is composed of a set of modules denoted by *Ms*, and the set of all modules in the cloud is *M* =_*s*∈*SMs*. A machine *n* ∈*N* in the cloud has a CPU capacity Ω*n* and a memory capacity Γ*n*. We use **Ω** and **Γ** to denote the vectors of CPU and memory capacities of all machines in the system. An instance of module *m* running on machine *n* demands *ωn,m*(*t*)CPU resource and *γm* memory resource from *n*. Machine *n* allocates to module *m* the CPU capacity ˆ*ωn,m*(*t*) and the memory capacity *γm*.By considering the memory Constriction we allocate the resource to virtual machine without considering the cost.

4.2. P* :A heuristic solution

P* is an asynchronous protocol. OP(2) consider the memory Constriction and minimized cost for job scheduling. This means that a machine does not synchronize the start time of a protocol round with any other machine of the cloud. At the beginning of a round a machine reads the current demands of the modules it runs. At the end of a round a machine updates its part of the configuration matrix *A*. The matrix *A* thus changes with dynamism and asynchronously during the evolution of the system.

P* employs the same basic mechanism as P': it attempts to equalize the relative demands of two machines during a protocol round. P* attempts to keep down the cost of reconfiguration by preferring not to start a new module instance during an equalization step op(2) in Figure 4.

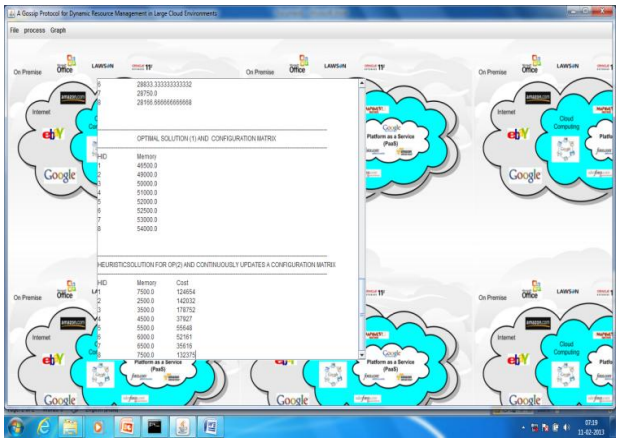

Figure 4.Heuristic solution to OP(2)

However, if a machine *n* performs equalization steps only with machines in *Nn*, there is a chance of partitioning the cloud into disjoint sets of interacting machines, which can result in a system state far from optimal. It attempts to equalize the relative demands of machines *n* and *n*_. It identifies the machine (i.e) host having total memory capacity and minimised cost for job scheduling.

Pseudo-code for heuristic solution

Step 1: Initially read CPU capacity Ω*n* and a memory capacity Γ*n*.
Step 2: Start the parallel processing threads (i.e.) active and passive threads.
Step 3: If active thread start to performing its job means then read CPU demand (ωn),memory demand (γn),configuration matrix (rown(A)),machine(Nn) (i.e.) read *ωn, γn, rown*(*A*),*Nn*.
Step 4: Randomly choose machine from Nn if rand(0..1) < p
Step 5: Else then choose randomly from the machine N – Nn
Step 6: While send function is invoke with parameters CPU demand (ωn),memory demand (γn),configuration matrix (rown(A)),CPU capacity (Ω*n*) to machine(Nn), send (*ωn, γn, rown*(*A*),Ω*n*) to *n'*. Parallel receive process take place in passive thread.
Step 7: Active thread invoke receive function send by passive thread , receive (*ωn' , γn', rown'* (*A*),Ω*n'*) from *n'* and then updated configuration matrix (*rown*(*A*)).
Step 8: Equalize function called to shift the demand with in the machine and finally after completion the task active thread move to sleep.
Step 9: If Passive thread receive CPU demand (ωn),memory demand (γn),configuration matrix (rown(A)),machine(Nn) to n machine send by active thread(i.e.) receive (*ωn' , γn', rown'* (*A*),Ω*n'*) from *n'*.
Step 10: Passive thread read(*ωn, γn, rown*(*A*),*Nn)* and send CPU demand, Memory demand, Configuration matrix to n' machine to active thread and finally update and write configuration matrix.
Step 11: Equalize function called to shift the demand with in the machine and finally after completion the task passive thread move to sleep.

4.3. MapReduce framework in heterogeneous environment

MapReduce builds on the observation that many information processing tasks have the same computational design computation is applied over a large number of web pages to generate partial results, which are then aggregated in some approach.MapReduce provides an abstraction for programmer designed mappers as "specifying per-record computations" and reducers as "specifying result aggregation" that both operate in parallel on key-value pairs as the processing primitives. The mapper is applied to every input key-value pair to generate an arbitrary number of intermediate key-value pairs. The reducer is then applied to all values associated with the same intermediate key to generate an arbitrary number of final key-value pairs as

output.This two-stage processing structure is illustrated in Figure 5. Under the MapReduce programming model, a developer needs only to provide implementations of the mapper and reducer. Mapper allows the execution framework transparently handles all other aspects of execution on clusters in Figure 6.

It is responsible for scheduling, handling faults and the large distributed sorting and shuffling problem between the map and reduce phases whereby intermediate key-value pairs must be grouped by key. As an optimization, MapReduce supports the use of "combiners" in Figure 5. which are similar to reducers except that they operate directly on the output of mappers; one can think of them as "mini-reducers". Combiners operate in seclusion on each node in the cluster and cannot use partial results from other nodes. Since the output of mappers must eventually be shuffled to the appropriate reducer over the network, combiners allow a programmer to aggregate partial results, thus reducing network traffic in Figure 7.

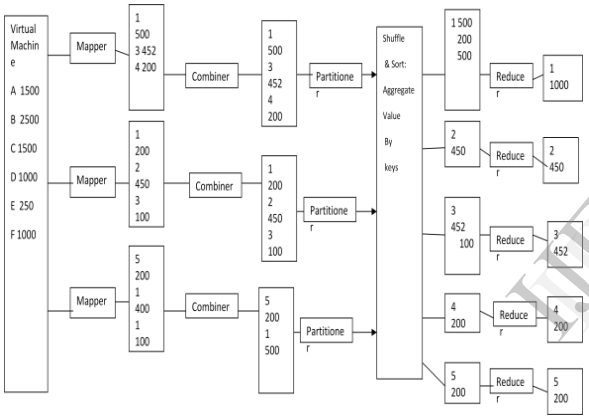

Figure 5. MapReduce illustration

In cases where an operation is both associative and commutative, reducers can directly serve up as combiners, although in general they are not interchangeable. The final component of MapReduce is the "partitioner" in Figure 5, which is responsible for dividing up the intermediate key space and assigning intermediate key-value pairs to reducers. The default partitioner computes the hash value of the key modulo the number of reducers. Partitioner shuffle and sort the intermediate aggregate values by keys in Figure 5,and generate cluster based on key in Figure 8, Partitioner value.

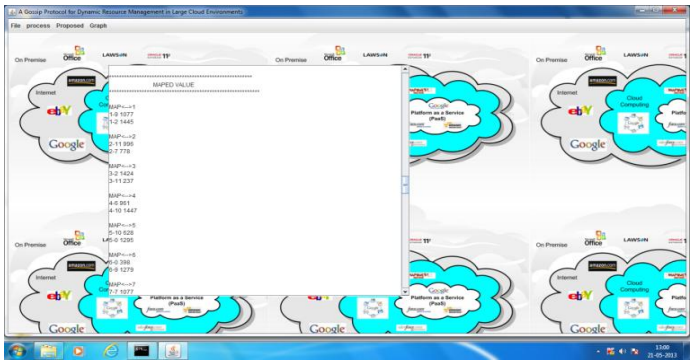

Figure 6. Cluster Formation by Mapper

Finally, reducer performed the aggregate function on partitioned value. It then applied to all values associated with the same intermediate key to generate an arbitrary number of final key-value pairs as output in Figure 8, Reducer value.MapReduce framework performed on program's execution across a set of machines, handling faults, and managing the required inter-machine communication are all handled by the run-time system. This enables programmers with no experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

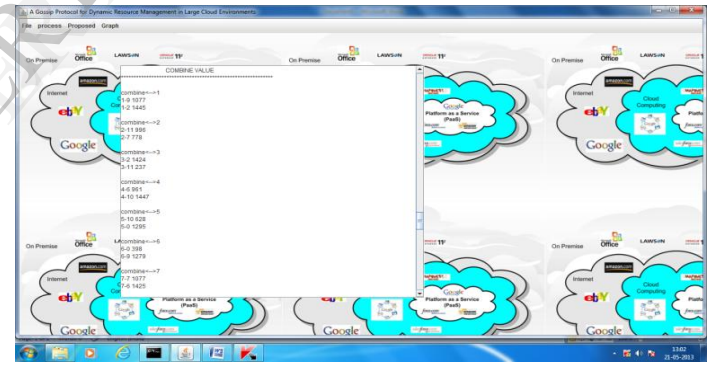

Figure 7. Combiner Function

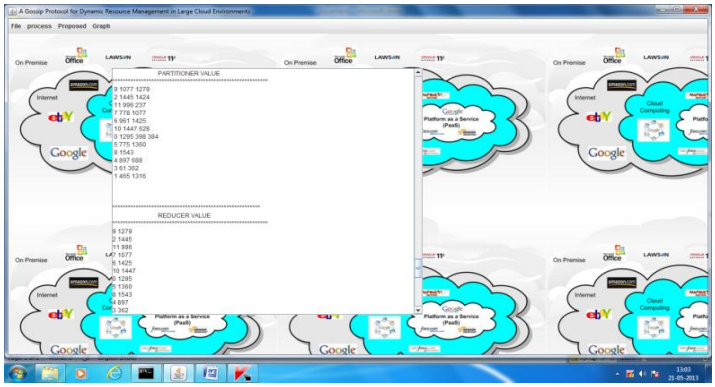

Figure 8. Partitioner and Reducer Value

Pseudo-code for MapReduce

Step 1: Initially the user specify the number of virtual machine, number of task and number of host to be created with in Cloud.
Step 2: Mapper form a cluster based on number of task divide by number of virtual machine. Each mapper will have map id, virtual machine id and bandwidth of task.
Step 3: Combiners operate in seclusion on each node in the cluster .It perform aggregate on each cluster by checking map id and virtual machine id.
Step 4: Partitioner which shuffle, sort and dividing up the intermediate key space and assigning intermediate key-value pairs to reducers. Separate cluster formed for each map id.
Step 5: Finally reducer perform the aggregate function on partitioned value.

5. Evaluation through simulation

We evaluate the MapReduce framework in heterogeneous cloud environment through simulations using a CloudSim simulator. CloudSim provides for P* the function of selecting a random machine for interaction. During Simulation we can specify the Number of Virtual Machine, Number of Tasks and Number of Hosts and freight of each site changes with dynamism with period and asynchronously. CloudSim is a new generalized and extensible simulation framework that enables seamless modelling, simulation, and experimentation of emerging Cloud computing infrastructures and management services. It support for modelling and instantiation of large scale Cloud computing infrastructure, including data centres on a single physical computing node and java virtual machine and flexibility to switch between space-shared and time-shared Share of processing cores to virtualized services.

Evaluation metrics: We run the protocol P* in various scenarios and measure the following metrics. Here op(1) represent Fairness resources Share in red line in homogeneous environment, op(2) represent Fairness resources Share with minimized cost in homogeneous environment in blue line and Map represent Fairness resources Share with minimized cost in heterogeneous environment in green line. We express the *sprite* of resource Share through the Coefficient of Variation of Fairness of resource allocated to number of task utilities. We measure the *satisfied demand* as the fraction of task that generates a utility less than 1. We measure the *cost of reconfiguration* as the number of new module instances started divided by the number of all module instances running at the end a sampling period, per machine and per sampling period.

Fairness : P* allocates CPU resources proportional to the demand of a module instance, regardless of the available capacity on the particular machine. The behavior is also to be expected, since OP(1) and op(2) need more memory Constriction to complete task in Figure 9. Maps have sufficient memory constraint for starting new instances. Fairness allocation efficiency is best in map function. Note that the ideal system always achieves optimal sprite, which means a value of 0.

Satisfied demand: The satisfied demand depends on both Memory constraints. For the ideal system, the satisfied demand depends only on CLF and hence is always equal to 1. In a situation where the CPU Share is fair all machines and allocated CPU resources that are less than their demand. For this value of op (1) and op (2), increasing MLF results in a more 'unfair' CPU Share. Since Map allocate fair share of resource that satisfy their demand in Figure 10.

Cost of reconfiguration: The cost of reconfiguration depend on memory constraint. The cost of reconfiguration can be

further reduced by controlling the trade off between achieving a higher utility vs. increasing the cost of a configuration in Figure 11.op(1) and op(2) required more cost compare to map in heterogeneous environment.

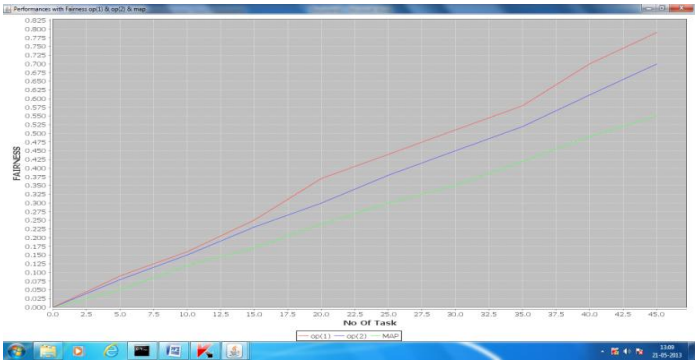

Figure 9.Fairness Share

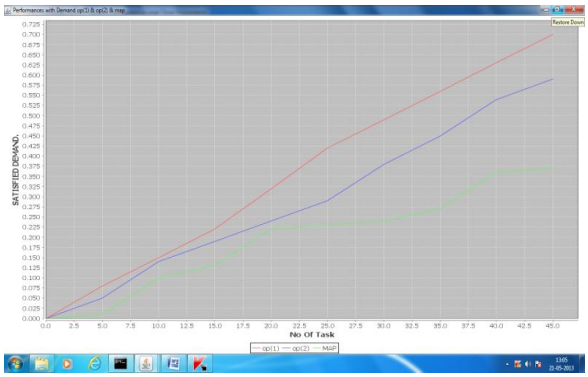

Figure 10.Satisfied Demand

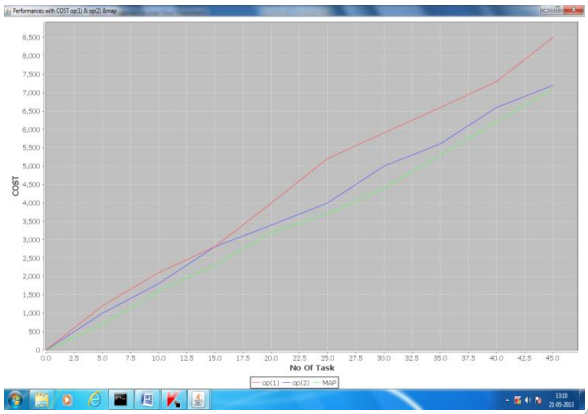

Figure 11.Cost of reconfiguration

6. Conclusion

We conclude that MapReduce framework in heterogeneous cloud environment that achieves high Concert metric and fairness resource Share. MapReduce which includes the mappers and reducers that both operate in similar as two stage processing on key-value pairs as the processing primitives. The job scheduler improves the input data locality of a virtual MapReduce cluster. With VM reconfiguration, each node can be adjusted to provide only the necessary amount of resource demanded for that node and efficient adaptation to load changes & scalability.Mapreduce results a metric of share in a heterogeneous cluster to realize a scheduling scheme that achieves high Concert and Fairness .

REFERENCES

[1] R. Yanggratoke, F. Wuhib, and R. Stadler, "Gossip-based resource Share for green computing in large clouds," in *2011 International Conference on Network and Service Management*.
[2] OpenStack LLC, http://www.openstack.org, Feb. 2012.
[3]http://www.ibm.com/software/webserver/appserv/virtualenterprise
/, Feb. 2012, "IBM Web Sphere Application Server,"
[4] VMware, http://www.cloudfoundry.com/, Feb. 2012.
[5] Amazon Web Services LLC, http://aws.amazon.com/ec2/, Feb. 2012.
[6] Google Inc., http://code.google.com/appengine/, Feb. 2012.
[7] Microsoft Inc., http://www.microsoft.com/windowsazure/, Feb. 2012.
[8] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregationin large dynamic networks," *ACM Trans. Computer Syst.*, vol. 23, no. 3,pp. 219–252, 2005.
[9] ——, "T-Man: gossip-based fast overlay topology construction," *ComputerNetworks*, vol. 53, no. 13, pp. 2321–2339, 2009.
[10] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource managementfor cloud Environments," in *2010 International Conference on Network and Service Management*.
[11] F. Wuhib, M. Dam, R. Stadler, and A. Clem, "Robust monitoring ofnetwork-wide aggregates through gossiping," *IEEE Trans. Network and Service Management*, vol. 6, no. 2, pp. 95–109, June 2009.
[12] F. Wuhib, M. Dam, and R. Stadler, "A gossiping protocol for detecting global threshold crossings," *IEEE Trans. Network and Service Management*,vol. 7, no. 1, pp. 42–57, Mar. 2010.
[13]Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in *2010 International Conference on Network and Service Management*.
[14] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility based placement of dynamic web applications with sprite goals," in*2008 IEEE Network Operations and Management Symposium*.
[15] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: in expensive membership management for unstructured p2p overlays," *J. Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.
[16] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J.Applied Mathematics*, vol. 17, no. 2, pp. pp. 416–429, 1969.
[17] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *2007 International Conference on World Wide Web*.
[18] H. Shachnai and T. Tamir, "On two class-constrained versions of the multiple knapsack crisis," *Algorithmica*, vol. 29, no. 3, pp. 442–467,
Dec. 2001.
[19] G. B. Dantzig, "Discrete-variable extremum crisiss," *OperationsResearch*, vol. 5, no. 2, pp. 266–288, 1957.
[20]C. Adam and R. Stadler, "Service middleware for self-managing largescalesystems," *IEEE Trans. Network and Service Management*, vol. 4,no. 3, pp. 50–64, Apr. 2008.
[21] J. Famaey, W. De Cock, T. Wauters, F. De Turck, B. Dhoedt, andP. Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in *2009 International Conference on Integrated Network Management*.
[22] C. Low, "Decentralised application placement," *Future Generation Computer Systems*, vol. 21, no. 2, pp. 281–290, 2005.
[23] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti,and Y. Coady, "Dynamic resource Share in computing clouds
using distributed multiple criteria decision analysis," in *2010 IEEEInternational Conference on Cloud Computing*.

[24] E. Loureiro, P. Nixon, and S. Dobson, "Decentralized utility maximizationfor adaptive management of shared resource pools," in *2009International Conference on Intelligent Networking and CollaborativeSystems*.
[25] Google Inc.,http://hadoop.apache.org

J.Keerthika has done BE (Computer Science Engineering), from Sri Subramanya college of engineering & technology, Palani affiliate to Anna University, Chennai in 2008. Currently doing her ME(Computer Science Engineering) final Year under Anna University Chennai, A.S.L.Pauls College of Engineering and Technology in Coimbatore. She published this paper in International conference on Innovations in Communication, Information and Computing (ICICIC'13) Sasurie College of Engineering, Tirupur, Tamil Nadu, India. January 9 -11, 2013, Published this paper in National Conference Computing Communication &Technology(N3CIT),T.J.S Engineering college, Chennai, Tamilnadu, India. April 10,2013. His research area is Distributed Management, Gossip protocol, Cloud Computing.

S.Castro received M.TECH from karunya University Coimbatore. Currently working as an Asst. Professor in Dept. of Computer Science Engineering, A.S.L.pauls College of Engineering and Technology, Coimbatore.